APPENDIX
IMPLEMENTATION DETAILS

The pseudocode of our method is Algo 1, and the implementation details of each component are in the following sections.

Algorithm 1 KUDA

```
Require: Vision-language Model $\mathcal{M}$, Dynamics Model $f$, Prompt Retriever $\mathcal{R}$, Point Tracker $\mathcal{T}$, Language Instruction $\mathcal{L}$, Text Prompt $p_{\text{base}}$, Prompt Library $\mathcal{P} = \{(q_i, \text{obs}_i, r_i)\}$
1: for high-level iteration $r = 0, \dots, N-1$ do
2:     Get observation $s_0$, environment state $z_0$ from the top down camera
3:     Propose keypoints and get the annotated image $A(s_0)$, use retriever $\mathcal{R}$ to get the top-k examples $\mathcal{R}(\{(q_i, \text{obs}_i, r_i)\})$
4:     Prompt the VLM to obtain the target specifications $\text{TS} = \mathcal{M}(p_{\text{base}}, A(s_0), \mathcal{R}(\{(q_i, \text{obs}_i, r_i)\}))$
5:     for low-level iteration $t = 0, \dots, \text{n\_actions} - 1$ do
6:         Obtain objective $\mathcal{C}(z|\mathcal{L}) = \sum_{i \in \text{TS}} \sqrt{(o_i - p_i)^2}$ from target specifications.
7:         Optimize $\mathcal{C}(z|\mathcal{L})$ with $f$ to get the action $a_t$
8:         Execute $a_t$ on the robot
9:         Obtain environment state $z_{t+1}$ and use tracking module $\mathcal{T}$ to update target specifications.
10:    end for
11: end for
```

A. Dynamics Models

We utilize two types of neural dynamics models: the graph-based neural dynamics model and the state-based neural dynamics model. The graph-based model follows the framework presented in [1]. Specifically, an object is represented as a graph with vertices $O = \{o_i\}$, where each vertex represents a particle, and edges $R = \{r_k\}$, which capture the relationships between particles. Each vertex o_i is defined as $o_i = \langle x_i, a_i^o \rangle$, where $x_i = \langle q_i, \dot{q}_i \rangle$ represents the state of particle i (including its position q_i and velocity $\dot{q}_i$), and a_i^o denotes its attributes. Each edge r_k is defined as $r_k = \langle u_k, v_k, a_k^r \rangle$, where u_k and v_k denote the receiver and sender vertices, respectively, and a_k^r represents the type and attributes of the relationship.

At each time step t, we employ two encoders, f_O^{enc} and f_R^{enc}, to compute latent embeddings for the vertices and edges, respectively:

$$h_{o_i,t}^0 = f_O^{\text{enc}}(o_{i,t}), \quad h_{r_k,t}^0 = f_R^{\text{enc}}(r_{k,t}). \tag{1}$$

Next, an edge propagation network f_R^{prop} and a vertex propagation network f_O^{prop} are used to iteratively update the embeddings of edges and vertices through multi-step message passing. For $l = 0, 1, \dots, L-1$, the updates are computed as:

$$h_{r_k,t}^{l+1} = f_R^{\text{prop}}(h_{u_k,t}^l, h_{v_k,t}^l), \tag{2}$$

$$h_{o_i,t}^{l+1} = f_O^{\text{prop}}\left(h_{o_i,t}^l, \sum_{j \in \mathcal{N}(o_{i,t})} h_{r_k,t}^{l+1}\right). \tag{3}$$

where $\mathcal{N}(o_{i,t})$ denotes the set of edge indices where vertex i acts as the receiver at time t, and L is the total number of message passing steps.

Finally, a vertex decoder f_O^{dec} is employed to predict the state of the object at the next time step, given the updated vertex embedding:

$$\hat{o}_{i,t+1} = f_O^{\text{dec}}(h_{o_i,t}^L). \tag{4}$$

To construct the graph $\{O, R\}$ in our approach, we utilize a top-down RGB-D camera to capture observations and utilize GroundingSAM [2] to segment objects and extract their corresponding point clouds. The farthest point sampling method is then applied with a fixed pointwise distance threshold r to generate particles representing the objects. For the two types of pushers used in our study, we represent the cylinder stick with a single particle and the board pusher with five particles. Edges between particles are established based on a spatial distance threshold d. In our implementation, we set $r = 0.02$ m and $d = 0.06$ m.

The data generation for our graph-based dynamics model training is achieved by NVIDIA FleX [1, 3], a position-based simulation framework tailored for modeling interactions involving various materials, including deformable objects. For each material, we collected a dataset comprising 1000 episodes, where each episode includes 5 randomly generated robot-object interactions. For the rope material, we randomize the length and stiffness of the rope, and for the granular material, we randomize the granular size, to enable our model to handle objects with different physical parameters. (You can see this in our demo, we used different kinds of rope and granular pieces.) To improve robustness during training, we incorporated rotational randomness into the simulation. Furthermore, our model is designed to be translation-equivariant, relying solely on velocity and position difference information within the network.

For T-shaped block in our work, we use four keypoints' x, y positions to represent T's state, which contains the top center point tc, top right point tr, top left point tl, and bottom center point bt when T is upright. At time t, the state-based neural network receives $(tc_t, tr_t, tl_t, bt_t, p_t, a)$ as input, where p_t is the current pusher position, and a represents the action of the pusher. Then the network will predict $(tc_{t+1}, tr_{t+1}, tl_{t+1}, bt_{t+1}, p_{t+1})$ for the next time step.

To obtain the keypoints of the T-shaped block in the real world, we first use the same pipeline as in the graph-based neural dynamics model to extract the point cloud of the T-shaped block. We then apply the Iterative Closest Point (ICP) algorithm [4] to estimate the block's 6D pose and calculate the positions of its keypoints. The T-shaped block used in our experiments measures 12 cm in height and 12 cm in width, with both the stem and bar having a width of 3 cm.

In our experiments, we employ different end effectors tailored to specific task requirements. The board pusher, used for manipulating cubes and granular pieces, has dimensions of 10 cm $\times$ 0.5 cm on the horizontal plane, whereas the cylinder pusher, designed for interacting with ropes and T-shaped objects, has a diameter of 1 cm.

The data generation for our state-based dynamics model is performed using Pymunk [5], a 2D physics library for simulating rigid body dynamics. For the T-shaped block, we collected a dataset of 20,000 episodes, with each episode consisting of 300 random robot-object interactions. During both training and inference, all coordinates are transformed into the block's local coordinate system, making our model both translation-equivariant and rotation-equivariant.

All training processes are performed on a Linux machine equipped with a CPU of 32 cores and 2 NVIDIA RTX 4090 GPUs.

B. Target Specification

We demonstrate the implementation details in Section **??** here. After capturing an RGB image using the top-down camera, we utilize SAM [6] to generate semantic masks for all objects in the scene. The mask with the largest area, which typically corresponds to the background, is discarded. For each remaining mask, we apply the farthest point sampling method with a fixed pointwise radius threshold to extract up to eight keypoints. Additionally, the center of each mask is included as a keypoint, as it often serves as a geometrically representative feature.

Subsequently, we apply the farthest point sampling method again, this time with a global radius threshold across all keypoints, to prevent excessive clustering near the edges. Each keypoint is marked on the original image as a red dot, with its index displayed above the dot. Additionally, a green dot is annotated at the center of the image and labeled as 'C', serving as a reference point when no other objects are present on the table.

All annotated points include both the keypoints on the objects and the reference points in the environment; they are not distinguished during annotation, as the VLM can typically recognize which points correspond to objects. We provide the example of our text prompt to the VLM in 1. Please see more detailed annotated image examples and text prompts in our code repository. We ensure that examples in the prompt library do not duplicate the objects and the instructions in evaluation tasks.

Listing 1: Text prompt for VLM

```
Please describe the final state of the object(s) on
     the table that satisfies the task by selecting
     keypoints and writing a Python function to
    specify their final positions.

The input request contains:
1. The task instruction describing what you are
    required to do.
2. An image of the current table-top environment
    captured from a top-down camera, overlayed with
     keypoints marked as P[i].

The response should be a Python function that
    describes the final spatial relationships
    between the keypoints of the object(s) you want
     to manipulate, and some other keypoints in the
     image.

The relationship is described by adding a 3D vector
     to the reference keypoint. For example, if P[i
    ], P[j] are two keypoints on the object, and P[
    a], P[b] are two other keypoints for reference,
     the function could be:

def keypoint_specification():
    p_i = p_a + [5, 0, 0]
    p_j = p_b + [0, 7, 0]
    return p_i, p_j

Imagine what the object(s) should finally look like
     after the task is completed, and select proper
     keypoints and describe their positions by
    referring to the near keypoints.

Note:
- x is left to right, y is bottom to top, z is from
     inside the image to outside the image, the
    unit is in cm.
- Please do not use variables in the 3D vector,
    follow the format p_i = p_a + [dx, dy, dz]. If
    there are no proper reference points on the
    table, you can also use p_i = [dx, dy, dz],
    while the origin is the center of the image,
    denoted as C.
- After your specification, a motion planner will
    match the chosen keypoints to their targets
    following an MSE loss.
- You can just specify several necessary keypoints
    to determine a pose instead of all the
    keypoints on the object(s) to make things
    easier.
- Here are the sizes of some possible items: the
    side length of the cube is 3cm, the L shape is
    9cm in width and 6cm in height, the rope is 40
    cm in length.
- Mention not to specify points that are not
    present in the image.
- If you think the task has been done, just return
    "Done."

Next I will show you some examples:
```

C. Dynamics Planning and Two Level Closed-loop Control

After projecting the target specifications into 3D space, as described in Target Specification, and obtaining the cost function, we apply the MPPI algorithm [7] to determine the next action. Specifically, starting from the initial environment state z_0, we iteratively sample actions $\{a_i\}_{i=0}^{T}$ from the action space, where T represents the look-ahead horizon. The dynamics model is then used to predict the outcome of each trajectory. Using the cost function, we calculate the weight of each trajectory and synthesize these trajectories to derive the final action sequence a_i that minimizes the cost function. In our setup, each action is a push along a straight line, with the starting point within the workspace and a length of no more than 20 cm.

To achieve closed-loop control at the dynamics planning level, we record a video of each action using a side camera. We then input all object particles, including keypoints from the target specifications, along with the recorded video into SpatialTracker [8] to obtain the positions of the tracked particles after each action. However, we observed that the tracked particles exhibit some errors, similar to the prediction errors from the dynamics model, making them unsuitable for the next optimization iteration. To address this, we update the target specifications by resampling the particles from the objects after each action. The tracked keypoints are then calibrated to their nearest neighbors among the newly sampled particles, and the target specifications and cost function are updated accordingly. Our experiment results demonstrate that this method effectively preserves the stability of the cost function.

To achieve closed-loop control at the VLM level, after a certain number of actions, we terminate the optimization process and update the current observation along with the language instruction to prompt the VLM in the next loop, generating a new target specification. Experimental results indicate that this method is particularly effective in under-specified tasks, where the number of keypoints is insufficient to accurately define a target for the objects. Additionally, it helps correct instances where the VLM provides incorrect target specifications.

REFERENCES

[1] Y. Li, J. Wu, R. Tedrake, J. B. Tenenbaum, and A. Torralba, “Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids,” in *ICLR*, 2019.

[2] T. Ren, S. Liu, A. Zeng, J. Lin, K. Li, H. Cao, J. Chen, X. Huang, Y. Chen, F. Yan, *et al.*, “Grounded sam: Assembling open-world models for diverse visual tasks,” *arXiv preprint arXiv:2401.14159*, 2024.

[3] M. Macklin, M. Müller, N. Chentanez, and T.-Y. Kim, “Unified particle physics for real-time applications,” *ACM Transactions on Graphics (TOG)*, vol. 33, no. 4, p. 104, 2014.

[4] S. Rusinkiewicz and M. Levoy, “Efficient variants of the icp algorithm,” in *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, 2001, pp. 145–152.

[5] V. Blomqvist, “Pymunk,” http://www.pymunk.org/, 2022.

[6] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, *et al.*, “Segment anything,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 4015–4026.

[7] G. Williams, A. Aldrich, and E. Theodorou, “Model predictive path integral control using covariance variable importance sampling,” *arXiv preprint arXiv:1509.01149*, 2015.

[8] Y. Xiao, Q. Wang, S. Zhang, N. Xue, S. Peng, Y. Shen, and X. Zhou, “Spatialtracker: Tracking any 2d pixels in 3d space,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 20 406–20 417.